Aquisição e Armazenamento de Metainformação no Contexto de um Arquivo

Miguel Ferreira
mferreira@dsi.uminho.pt
DSI/UM

José Carlos Ramalho
jcr@di.uminho.pt
DI/UM

7 de Fevereiro de 2004

Resumo

Neste artigo, apresentam-se as várias etapas que levaram à construção dum repositório de metainformação no contexto de um arquivo histórico digital.

O XML surge naturalmente neste projecto pois a comunidade arquivística internacional adoptou há já algum tempo a linguagem de anotação EAD (*"Encoded Archival Description"*) para a descrição de metainformação arquivística e todo o intercâmbio de informação entre os vários actores (arquivos, tribunais, notários, paróquias, e outros) é feito em XML segundo a norma EAD.

Apesar do EAD existir no domínio arquivístico há alguns anos, este projecto encerrou alguns desafios interessantes. Em primeiro lugar foi preciso verificar de que maneira se poderia aplicar o EAD à realidade portuguesa. Em segundo lugar, o EAD segue uma estrutura hierárquica, natural no XML e natural também na descrição arquivística, mas o modelo a implementar teria de ser relacional. Assim houve que criar um modelo relacional que reflectisse a hierarquia da informação que seria armazenada.

Ao nível da aplicação de gestão de metainformação houve também que resolver o mesmo problema, pretendia-se manipular hierarquicamente informação que estava armazenada relacionalmente.

Estes e outros pontos satélites, como sendo a aquisição de metainformação e a transformação dos sistemas legados existentes, são discutidos no artigo.

1 Introdução

Quem já visitou, e se deteve pacientemente no serviço de referência de um Arquivo Histórico, sabe a confusão com que normalmente se depara. Múltiplos índices, livros de listagens, inventários, catálogos e guias de transferência que foram sendo elaborados ao longo do tempo, fruto de valioso trabalho mas, apesar disso, multiformes e sem coerência colectiva. Para pôr um fim definitivo a este cenário, foi desenvolvido no Arquivo Distrital do Porto (ADP) o projecto DigitArq, um projecto com múltiplas frentes, mas com um objectivo fundamental: servir de primeira abordagem à edificação de um *Arquivo Digital*.

Uma das componentes basilares deste projecto consistiu na conversão de materiais descritivos existentes somente em papel para formatos digitais normalizados baseados em normas internacionais. Assim, após a digitalização dos materiais, recorreu-se ao auxílio de software de reconhecimento óptico de caracteres (OCR) de forma a obter texto bruto capaz de ser tratado digitalmente. Após esta fase, o contributo de técnicos especializados em arquivística foi fundamental, de forma a corrigir alguns erros residuais da fase de digitalização, e para auxiliar em todo o processo de anotação do texto resultante. Uma vez o texto anotado, a sua conversão para formatos XML normalizados foi trivial. Diversos scripts transformadores foram desenvolvidos, que quando aplicados aos diferentes documentos resultaram em XML baseado na norma EAD (Encoded Archival Description).

Outra das componentes deste projecto consistiu na retro-conversão de bases de dados, agora obsoletas ou desadequadas, para um modelo de dados baseado na norma EAD. A título de exemplo, podemos mencionar a existência de materiais digitais armazenados em documentos Word, Excel, Access, XML, Arqbase/ISIS, etc.

Após as conversões descritas foi detectada uma quantidade assinalável de material em formato EAD. Tornou-se, pois, essencial, armazenar as várias centenas de ficheiros resultantes da migração num repositório de informação centralizado que permitisse, não só, instalar organizadamente esses ficheiros, como também, permitir o acesso à informação neles contida. Uma ferramenta que satisfizesse esses requisitos, mas que também, assistisse o operador

na produção e manutenção de novas descrições arquivísticas tornou-se assim necessária. Nesse sentido, foi criada uma aplicação de descrição arquivística que reúne um número considerável de funcionalidades com o objectivo de facilitar e apoiar a produção de descrições normalizadas.

Ao longo deste artigo descreve-se com mais detalhe as várias fases do projecto. Na próxima secção (sec. 2), tenta caracterizar-se o contexto em o XML surge neste projecto. Na secção seguinte (sec. 3), discutem-se alguns requisitos dos futuros utilizadores que tiveram um forte impacto na solução proposta. A secção 4 é onde se apresenta o modelo desenvolvido para suportar o repositório de metainformação. Nesta secção, também se apresenta a interface criada para manipular o modelo de dados. Na secção 5 discute-se a implementação da aplicação e para terminar o artigo (sec. 6), apresentam-se algumas conclusões e pontos ainda por resolver.

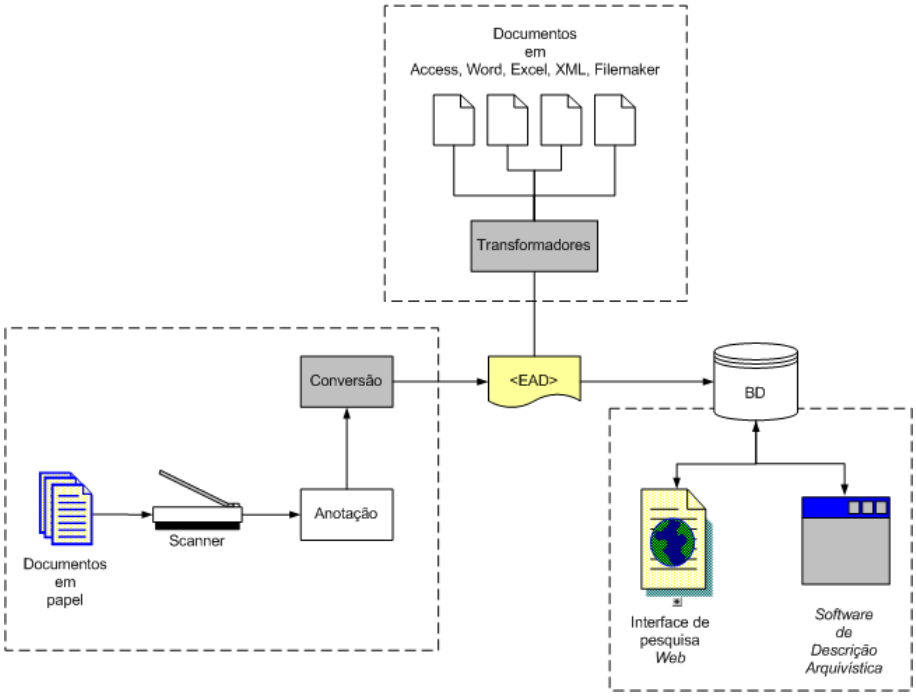

Figura 1: Diferentes fases do projecto.

Sobre o repositório de dados, foi também desenvolvido um motor de pesquisa acessível via *Web*. Uma ferramenta de trabalho indispensável aos utentes do Arquivo, especialmente àqueles que residem além-fronteiras e que não possuem disponibilidade para se deslocarem fisicamente às instalações do Arquivo.

2 O XML ao longo do projecto

Um dos principais requisitos do projecto DigitArq, para além da recuperação e centralização de toda a informação que se encontrava distribuída por diversas bases de dados, assentava na necessidade dessa mesma informação ter de ser trocada com outros repositórios nacionais e internacionais. A norma Encoded Archival Description (EAD) foi escolhida para desempenhar essa função por se basear em XML, cujas vantagens neste contexto são amplamente conhecidas, e por se estar a tornar numa norma *de facto* no meio arquivístico para armazenamento e estruturação de informação. Além disso, assegura a criação de descrições consistentes, apropriadas e auto-explicativas e possibilita a partilha de dados de autoridade que tornam possível a integração de descrições de diferentes arquivos num sistema unificado de informação.

A informação arquivística é portadora de algumas características particulares. Para começar, encontra-se normalmente organizada hierarquicamente, efectuando uma descrição do mais geral para o mais particular. Assim, um documento alojado num Arquivo nunca se encontra descrito isoladamente (como acontece com um livro de uma biblioteca). Existe sempre uma relação entre o documento descrito e a entidade que o produziu, bem como uma descrição de todas as divisões e subdivisões dessa mesma entidade. Podemos, portanto, considerar uma descrição

arquivística como uma árvore cujos nós descrevem diferentes partes de uma mesma organização. Ao nível da raiz possuímos a descrição do fundo (a organização que gerou o documento) e nos restantes níveis, a descrição das diferentes partes que o compõem. Ao nível das folhas são descritos os documentos simples, ou seja, aqueles que não podem ser progressivamente subdivididos em mais níveis de descrição.

As vantagens do uso de XML neste contexto são óbvias, pois o armazenamento hierárquico da informação está automaticamente assegurado. No entanto, o XML possui um conjunto de características que o tornam difícil de manusear:

1. *O XML é baixo nível*
   Quer queiramos quer não, o XML é demasiado baixo nível para poder ser manipulado directamente por um utilizador. Os DTDs crescem de uma forma assustadora, de maneira que, é difícil encontrar uma norma internacional que não possua, pelo menos, várias centenas de elementos. Assim, é do interesse de todos, que o XML se mantenha, a toda a hora, escondido do utilizador comum, recorrendo a interfaces gráficas amigáveis que impossibilitem a ocorrência de erros na sua sintaxe.

2. *O XML é difícil de armazenar*
   Existe, hoje em dia, uma panóplia de opções no que diz respeito ao armazenamento de XML. O mercado das bases de dados com suporte para XML está em constante crescimento. Não obstante, podemos distinguir três estratégias fundamentais para o armazenamento de XML:

   (a) Sistemas XML nativos

   (b) Extensões XML (a sistemas RDBMS e OODBMS já existentes)

   (c) Sistemas XML virtuais (baseados em middleware entre as aplicações e o DBMS)

No entanto, não existe consenso no que diz respeito à melhor estratégia a seguir. Uma análise detalhada dos requisitos da aplicação terá que ser efectuada de forma a determinar qual das aproximações se adequa mais eficazmente ao cumprimento desses mesmos requisitos. Um estudo comparativo das diversas alternativas, realizado por R. Nunes e M. Silva [Nun03] conclui que, as bases de dados XML nativas, embora muito rápidas na recuperação da informação, consomem demasiado tempo durante a indexação de documentos de tamanho considerável e o uso de modelos relacionais, dotados de extensões, apenas são adequados quando a estrutura do XML é previamente conhecida e bem definida por um DTD.

3 Interface gráfica para descrição arquivística

A informação produzida por um Arquivo baseia-se, na sua essência, em meta-informação sobre os documentos albergados e as organizações que os produziram. Descrições intermédias entre ambos permitem catalogar os documentos no contexto da organização que os produziu, como por exemplo, identificar qual o serviço ou sucursal que gerou um documento específico.

Segundo as boas regras da arquivística, cada registo, ou nó, dessa árvore de descrição encontra-se identificado por um código de referência. Assim, um qualquer registo pode ser univocamente identificado pela concatenação das respectivas referências, desde o nível raiz até ao documento em causa. Por exemplo, a referência completa EMP-BM/L/001/00001, poderá ser interpretada como pertencendo ao fundo EMP-BM (Empresa Banco do Minho), subfundo L (sucursal de Lisboa), série 001 (correspondência recebida) documento 00001 (a referência do documento propriamente dito).

Todos os sistemas de descrição arquivística que analisamos baseavam-se neste conceito de referências completas para identificar a posição da árvore onde o registo deveria estar pendurado. As interfaces gráficas de introdução de dados eram, assim, baseadas num único formulário onde todos os campos de meta-informação podiam ser introduzidos e onde um dos campos a preencher consistia na referência completa do registo. A referência do registo servia, assim, tanto para identificar o registo como para o situar na árvore de descrição.

Acontece, no entanto, que a ocorrência de erros aquando da introdução das respectivas referências (pelos operadores) é frequente, fazendo com que um registo salte para uma posição da árvore completamente distinta daquela que era pretendida, ou pior ainda, que o registo nem sequer possua significado na árvore em questão. Para além disso, o esforço mental que um operador necessita de fazer para visualizar a estrutura de uma organização é tremendamente desgastante, sendo este um dos principais causadores dos frequentes enganos.

De forma a minimizar a ocorrência de erros que deterioram francamente a qualidade das descrições, propusemos a realização de uma interface gráfica que representasse visualmente a árvore de descrição eliminando a necessidade de introdução de longas referências e impedindo o operador de cometer erros aquando da sua introdução. Assim, a interface gráfica está dividida em duas áreas distintas, uma constituída por uma árvore representativa do fundo em que se está a trabalhar e uma outra onde são apresentados os campos que podemos preencher para descrever o registo seleccionado (figura 2).

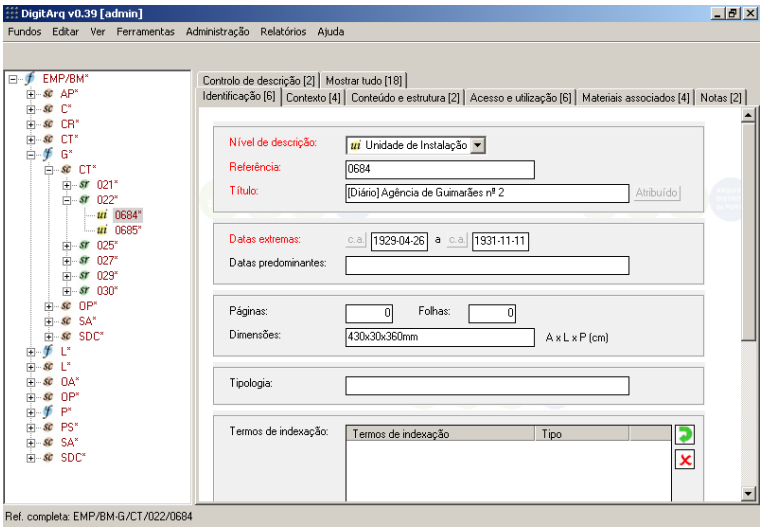

Figura 2: Interface do software de descrição arquivística.

Em teoria arquivística, cada registo encontra-se posicionado num nível de descrição, uma espécie de catalogação que caracteriza o tipo de registo. No contexto do Arquivo Distrital do Porto foram identificados dez níveis de descrição distintos: Fundo, Subfundo, Secção, Subsecção, Subsubsecção, Série, Subsérie, Unidade de instalação, Documento composto e Documento simples. A interface gráfica seria, também, capaz de garantir a coerência da descrição, impedindo o utilizador de desrespeitar a lógica hierárquica inerente, e.g., a interface não deveria permitir a criação de uma Secção debaixo de uma Subsecção, pois isso violaria a lógica hierárquica subjacente. Além disso, o software deveria ser capaz de detectar incoerências e omissões na descrição elaborada. Existem campos que são considerados obrigatórios, como a Referência, o Título ou as Datas extremas. O programa deveria alertar o utilizador quando algum destes campos não fosse preenchido, ou a informação nele contida fosse incoerente, e.g., uma data final superior a uma data inicial.

Para além dos cuidados a ter com a qualidade da descrição, o software deviria ser possuidor de algumas características adicionais, como por exemplo, permitir a inferência de informação de níveis inferiores para níveis de topo. As datas extremas associadas a um registo de nível fundo deverão ser, respectivamente, a data inicial do primeiro documento produzido pela organização e a data final o documento mais antigo. Assim, através de métodos de inferência deveria ser possível transportar para níveis superiores informação relativa aos documentos folha da árvore.

O software de descrição deveria, também, ser capaz de permitir que vários operadores pudessem trabalhar simultaneamente sobre um mesmo fundo, podendo observar em tempo real as alterações efectuadas pelos outros operadores, obtendo assim uma visão geral do todo o trabalho concretizado pela equipa de descrição.

4 Modelo de dados

No projecto DigitArq foi adoptada uma abordagem *top-down*, tomando como ponto de partida a interface gráfica que desejávamos desenvolver e analisando todos os requisitos que a mesma deveria suportar. A partir daí, construímos um modelo de dados capaz de suportar todas exigências previamente definidas. Após uma análise superficial da norma EAD e da constatação da sua complexidade passamos à fase que normalmente antecede a utilização de uma qualquer norma baseada em XML - a *adaptação da norma*. Esta, consiste na análise de requisitos do sistema a desenvolver e no teste de adequabilidade da norma a utilizar, recorrendo, por vezes, à adaptação de alguns

elementos para suportar a informação pretendida e à rejeição de elementos desnecessários. A fase de adaptação da norma é fundamental ao desenvolvimento de qualquer projecto baseado em XML, pois transforma uma pesada norma internacional numa estrutura de tratamento simples e de tamanho manejável adequada às necessidades específicas dum determinado projecto.

4.1 Adaptação do EAD

Esta secção pretende servir de guia para a codificação de auxiliares de pesquisa em EAD/XML. Foi nossa intenção excluir deste documento uma introdução à norma EAD. Existe, na Internet, bastante documentação sobre este assunto [oAA02]. Assume-se, portanto, que o leitor conhece a norma EAD e que está a par das suas ambiguidades. A norma é bastante liberal na maior parte das suas componentes. Assim, algumas decisões tiveram que ser tomadas de forma a conseguir codificar a informação extraída, das diversas bases de dados.

De seguida são descritas algumas das decisões tomadas no âmbito do projecto DigitArq.

4.1.1 Datas extremas

Um dos primeiros problemas detectados na norma EAD consistia na inexistência de elementos capazes de acolher as datas extremas dos registos retro-convertidos. Por norma, um documento é portador de duas datas extremas, a data de criação inicial e final. Em alguns casos essas datas poderão coincidir. Um caso típico de utilização de datas extremas ocorre geralmente quando descrevemos livros. Um livro de baptismos, por exemplo, possui, ao longo das suas páginas, centenas de registos, fazendo com que as datas extremas do livro sejam, respectivamente, as datas do primeiro, e do último baptismo registados.

A norma EAD apenas contempla a existência de um elemento para representar datas extremas, sendo da responsabilidade do utilizador assegurar a coerência do seu formato. Existem diversas soluções para este problema, no entanto, todas estas recaem sobre dois modelos fundamentais: utilizar uma sintaxe bem definida para as datas que permita distinguir a data inicial da data final, ou utilizar um atributo para indicar a qual das datas o elemento descritivo se refere.

No primeiro caso, se quiséssemos representar as datas extremas da crise cerealífera que assolou o nosso país no século XV, poderíamos fazê-lo recorrendo à notação `<unitdate>1436/1441<unitdate>`.

No segundo caso, a representação seria assegurada pelos elementos `<unidate datechar='inicial'> 1436 </unitdate>` e `<unidate datechar='final'> 1441 </unitdate>`, onde o atributo `datechar` indica a que data o elemento `unitdate` pretende descrever.

Em ambos os casos, a sintaxe escolhida para representar as datas deverá ser bem definida, pois é fundamental que se possam efectuar pesquisas sobre as datas dos registos, tendo obviamente em consideração os intervalos definidos.

Outro problema, que acompanha a codificação de datas, tem que ver com a representação de informação incompleta, ou seja, como representar as datas para as quais não conhecemos, por exemplo, o dia ou o mês. O formato escolhido deve permitir determinar se uma qualquer data pertence a um intervalo definido por duas datas incompletas.

No que diz respeito às datas, as decisões tomadas podem ser resumidas da seguinte forma:

- *Utilização de atributos para identificar a extremidade da data*
  Evita a necessidade de processamento adicional para separar as datas na eventualidade de ser necessário efectuar qualquer tipo de operação com as mesmas.

- *As datas são sempre representadas no formato YYYY-MM-DD*
  Garante a uniformidade das datas e simplifica todo o tipo de cálculos a efectuar com as mesmas.

- *Informação incompleta*
  A informação incompleta é representada por uma cadeia de zeros com o mesmo comprimento do elemento em causa, e.g., na data 1436-04-00 depreende-se que o dia é desconhecido.

4.1.2 Numbered vs Unnumbered Components

Em EAD cada nível de descrição pode ser representado de duas formas: utilizando os elementos <c1>, <c2>, ...,<c12>, cujo valor numérico associado representa a profundidade do elemento na árvore, ou utilizando um elemento não numerado simplesmente representado por <c>.

No âmbito do nosso projecto não sentimos necessidade de utilizar os elementos numerados, pois a profundidade da hierarquia é automaticamente descrita pela disposição dos diversos elementos no documento XML. Optamos assim, pela utilização do elemento <c> uma vez que o tratamento numérico da profundidade, para além de supérfluo, acarreta processamento adicional.

Nos exemplos que se seguem, ambas as representações são perfeitamente equivalentes:

```
<c01>
    <c02>
        <c03>
            ...
        </c03>
    </c02>
    <c02>
        ...
    </c02>
</c01>
```

Exemplo 1: Anotação usando *Numbered Components*.

```
<c>
    <c>
        <c>
            ...
        </c>
    </c>
    <c>
      ...
    </c>
</c>
```

Exemplo 2: Anotação usando *Unnumbered Components*.

É de notar que o elemento <c> (*component*) representa um nó da árvore de descrição documental. Pendurados neste elemento, encontram-se todos os elementos que carregam informação útil produzida pelo arquivista, e.g., o código de referência e o título do elemento descritivo.

Um registo cujo código de referência é EMP-BM e cujo título é Banco do Minho podem ser codificados em EAD da seguinte forma:

```
...
<c>
    ...
    <unitid>EMP-BM</unitid>
```

```
    <unittitle>Banco do Minho<unittile>
    ...
</c>
...
```

4.1.3 Atributo otherlevel

Associado a cada elemento `<c>` (*component*) existe um atributo `level` que indica qual o nível de descrição arquivística que o elemento representa. No seio do nosso projecto foram definidos os níveis de descrição: Fundo, Subfundo, Secção, Subsecção, Subsubsecção, Série, Subsérie, Unidade de instalação, Documento composto e Documento simples. O atributo `level` do elemento *component* descrito pela norma EAD/XML define os níveis de descrição: class, collection, fonds, subfonds, series, subseries, file, item, recordgrp , subgrp e otherlevel.

Como podemos constatar, os níveis de descrição definidos na norma não satisfazem as necessidades do Arquivo Distrital do Porto. Assim, foi utilizado o atributo opcional `otherlevel` onde pudemos descrever o nosso nível de descrição, bastando para isso identificar o nível de descrição como sendo `otherlevel`.

Exemplo: `<c level='otherlevel' otherlevel='Fundo'> ...</c>`

4.1.4 Elementos mistos

O EAD contempla a utilização de alguns elementos especiais para anotar secções de texto no interior de outros elementos que se designam por elementos de conteúdo misto: os chamados elementos flutuantes. Assim, é possível, por exemplo, na descrição do Âmbito e Conteúdo, de um qualquer registo, assinalar que um determinado conjunto de palavras diz respeito ao nome de uma pessoa.

```
<scopecontent>
  O fundador, <person>Manuel Ferreira</person>, constituiu ...
</scopecontent>
```

A utilidade dos elementos *flutuantes* neste contexto é amplamente reconhecida, no entanto, a sua implementação do ponto de vista da interface gráfica acarreta alguns problemas. Como poderemos permitir a anotação de texto, sem que as etiquetas sejam visualizadas? Usando um sistema de cores para diferenciar o texto anotado? Seria uma hipótese, no entanto, se encararmos o problema do ponto de vista relacional veremos que é demasiado complexo de tratar. Assim, todos os elementos *flutuantes* foram substituídos por termos de indexação. Por outras palavras, acabaram-se com os conteúdos mistos (estes são normalmente o problema quando se pretende guardar documentos num modelo relacional).

4.1.5 Termos de indexação

A linguagem natural é utilizada correntemente em todas as actividades da vida quotidiana. De acordo com o contexto, o nível da língua será familiar, erudito, técnico, poético, literário, diplomático, administrativo, etc. Vários termos são sinónimos, antónimos; são específicos ou genéricos. Esta diversidade ilustra a riqueza e a subtileza da língua. Geralmente, os títulos das obras são redigidos em linguagem natural, o que facilita a sua compreensão pelos leitores.

Na linguagem natural, pode-se escolher um termo, ou um conjunto de termos para se fazer uma descrição, uma representação, uma ilustração ou uma síntese de um assunto, de uma ideia, de um texto, de uma situação.... etc. O termo assim escolhido torna-se uma palavra-chave. Esta palavra-chave permite-nos identificar o que é importante ou o que é preciso reter para compreender o essencial de um assunto. Consoante o caso, a palavra-chave tem um sentido geral ou específico.

A utilização da palavra-chave ou de um conjunto de palavras-chave, é uma estratégia eficaz para delimitar um assunto de pesquisa. Também nesta perspectiva, quanto mais a palavra-chave for precisa e pertinente mais fácil será o início da pesquisa da informação. Além disso, a utilização da palavra-chave torna-se essencial para fazer a ligação com as linguagens especializadas dos diversos domínios do conhecimento humano. A linguagem controlada utiliza termos escolhidos, precisos e unívocos com a finalidade de evitar as interpretações de sentido, permitindo assim a indexação precisa dos registos [dRdBE03].

Assim, associado a cada registo, existe uma lista de palavras-chave assentes em linguagem controlada, que permitem identificar os principais tópicos abordados. Como a linguagem é bem definida, é possível efectuar pesquisas eficazes sobre a meta-informação existente.

No projecto DigitArq os termos de indexação foram hierarquizados a dois níveis. O primeiro identifica uma série de categorias às quais podemos adicionar termos concretos. Esta opção reduz a ambiguidade inerente a algumas palavras ou expressões. Por exemplo, será que ”Elias Garcia”diz respeito ao nome de uma rua ou ao nome de uma escola secundária de Almada? Ao descrevermos o termo como ”Toponímia/Elias Garcia”concluímos imediatamente que se trata do nome de uma rua. Em EAD o exemplo representar-se-ia da seguinte forma:

```
...
<controlaccess>
    <list>
        <defitem>
            <label>Toponímia</label>
            <item>Elias Garcia</item>
        </defitem>
    </list>
</controlaccess>
...
```

A definição dos termos a utilizar fica a cargo do administrador do sistema, ou seja, alguém no interior do Arquivo que possui autoridade para gerir a lista de termos de indexação.

4.2 XML vs SQL

As diversas fases de conversão resultaram em milhares de registos migrados de diversas bases de dados. O novo sistema de informação deveria garantir que toda a informação convertida fosse incorporada e tornada acessível ao público geral. Tornou-se, assim, fundamental a criação de um repositório central para que fosse possível gerir toda essa informação, bem como, permitir a realização de pesquisas através de uma interface Web. Para a realização dessa tarefa foi utilizada uma base de dados relacional.

A escolha de uma base de dados relacional recaiu, em parte, nas condicionantes orçamentais para a aquisição de uma base de dados XML nativa e nas restrições temporais vigentes, que impediam que demasiado tempo fosse dispendido em instalação e configuração.

Foi necessário desenvolver um modelo de dados capaz de funcionar sobre uma base de dados relacional que conseguisse reflectir, dentro do possível, a informação contida nos ficheiros EAD/XML resultantes das diversas conversões. A transição de um dos modelos para o outro (XML/Relacional) deveria ser simples e transparente. Foi então desenvolvida, uma camada intermédia de software para funcionar entre a aplicação de gestão, e a base de dados.

A camada intermédia (ou *middleware*) é descrita por uma classe abstracta onde são definidos nove métodos fundamentais:

- `Sub Upload()`
  Actualiza a informação do nível de armazenamento com a informação residente em memória.
- `Sub Download()`
  Actualiza a memória com a informação do nível de armazenamento. Este método permite refrescar os dados em memória de modo a garantir que a interface apresenta a versão mais recente do registo.
- `Function Children() As LazyNodeCollection`
  Retorna uma colecção com os nós filhos. Se não existirem filhos retorna uma colecção vazia.
- `Sub RemoveChild(ByVal child As LazyNode)`
  Remove um filho do nó actual.
- `Sub AppendChild(ByVal child As LazyNode)`
  Adiciona um novo filho ao nó actual.

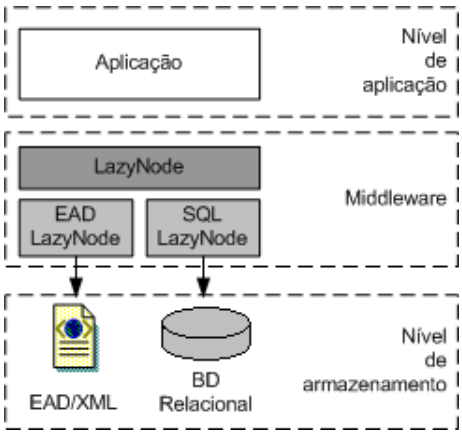

Figura 3: Diagrama de blocos.

Propriedade	**Elemento EAD/XML**
Otherlevel	@otherlevel
Unitid	did/unitid
CountryCode	did/unitid/@countrycode
RepositoryCode	did/unitid/@repositorycode
...	...

Tabela 1: Relação entre propriedades do LazyNode e elementos do EAD/XML

- `Function HasChildren() As Boolean`
  Retorna verdadeiro se o nó actual tiver filhos, e falso em caso contrário.
- `Function CreateNode() As LazyNode`
  Cria um novo nó. O novo nó terá que ser adicionado como filho ao nó pretendido.
- `Function Clone() As LazyNode`
  Cria uma cópia do nó actual.
- `Function Parent() As LazyNode`
  Retorna o pai do nó actual. Caso não exista (o nó actual é a raiz) retorna o valor nulo.

Para além destes métodos, que cada implementação concreta da classe abstracta é obrigada implementar, cada nó carrega consigo um conjunto de propriedades que permitem conservar em memória os valores de cada campo de informação que o nosso sistema é capaz de manipular.

A classe abstracta foi baptizada de LazyNode (nó preguiçoso), uma vez que, a informação apenas é transportada do nível físico para a memória, a pedido. Desta forma não sobrecarregamos a memória do sistema com informação indesejada.

Foram desenvolvidas duas implementações da classe LazyNode: SQLLazyNode e EADLazyNode. Cada uma das implementações limita-se a implementar os nove métodos herdados.

4.3 EADLazyNode

A classe EADLazyNode opera sobre documentos EAD/XML fazendo uso do DOM (W3C Document Object Model) como base de suporte para a manipulação dos ficheiros XML. Assim, todas as operações descritas anteriormente são implementadas com métodos disponibilizados pelo DOM.

Existe uma relação directa entre cada propriedade e a sua representação em XML (tabela 1).

As operações de leitura sobre um documento XML limitam-se a retornar o valor do elemento indicado pelo XPath. Caso o elemento não exista, é retornado o valor nulo.

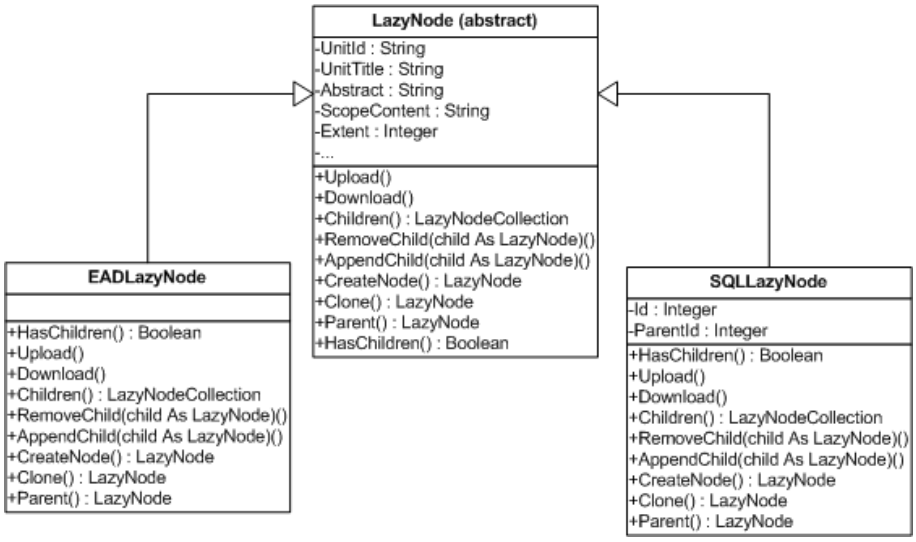

Figura 4: Diagrama de classes do LazyNode.

Antes de uma operação de escrita é verificada a existência do XPath correspondente à propriedade que se pretende escrever. Caso não exista, o caminho é criado e o valor do respectivo elemento é actualizado.

5 SQLLazyNode

A implementação do SQLLazyNode permite-nos manipular a informação quando esta se encontra armazenada numa base de dados relacional. Cada nó da árvore de descrição documental (*component* <c>) é descrito, no contexto relacional, por um registo que contém uma coluna por cada propriedade definida. Para além das propriedades, foi necessário adicionar alguns campos de controlo: Id, ParentId e HasChildren.

O modelo hierárquico inerente à árvore de descrição documental é assegurado por uma relação circular, onde o campo ParentId de um registo aponta para o Id do registo hierarquicamente superior (figura 5). Um registo raiz possui um apontador nulo (ParentId = NULL), ou seja, não possui pai.

Foi utilizado um campo adicional de controlo designado *HasChildren* (boleano) para indicar se um determinado nó da árvore possui, ou não, filhos. Assim, é possível obter esta informação sem sobrecarregar a base de dados com uma consulta demasiado pesada.

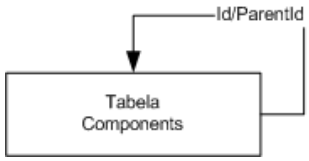

Figura 5: Diagrama entidade-relação.

As operações obrigatórias são implementadas de uma forma simples sobre o modelo relacional:

- `Sub Upload()`

```
UPDATE Components
SET {campos a actualizar}={valor}
WHERE ID={Id do nó actual}
```

- `Sub Download()`

```
SELECT *
FROM Components
```

- `Function Children() As LazyNodeCollection`

```
SELECT Id
FROM Components
WHERE ParentId={Id do nó actual}
```

- `Sub RemoveChild(ByVal child As LazyNode)`

```
For Each Node In Child.Children
    RemoveChild(Node)
Next
Dim SQLChildNode As SQLLazyNode = CType(Child, SQLLazyNode)

DELETE FROM Components
WHERE ID = {Child.Id}
If Children.Count = 0 Then
    UpdateHasChildren(False) 'updates the HasChildren Flag
End If
```

- `Sub AppendChild(ByVal child As LazyNode)`

Assumindo que foi executado previamente um CreateNode() ou um Clone() de modo a obtermos uma referência para um novo nó:

```
UPDATE Components SET ParentID={Id do nó actual}
WHERE ID={Id do nó a adicionar}
```

- `Function HasChildren() As Boolean`

```
SELECT HasChildren
FROM Components
WHERE ID = {Id do nó actual}
```

- `Function CreateNode() As LazyNode`

```
INSERT INTO Components ({campos não nulos})
VALUES ({valores por omissão})
```

- `Function Clone() As LazyNode CreateNode()`

```
NewNode.ImportFields(Me)
NewNode.Upload()

Dim Child As LazyNode For Each Child In Children()
    Dim NewChild As LazyNode = Child.Clone
    NewNode.AppendChild(NewChild)
Next
```

- `Function Parent() As LazyNode`

```
SELECT ParentID
FROM Components
WHERE Id={Id do nó actual}
```

5.1 Vantagens e desvantagens do modelo adoptado

O modelo apresentado é simples e apresenta algumas características interessantes. A primeira de todas é a capacidade podermos realizar catamorfismos sobre o modelo descrito sem termos que nos preocupar com as particularidades do nível de armazenamento. Esta característica é realmente importante se considerarmos que a maior parte das operações desempenhadas pelo software de gestão documental são realizadas por fundo (árvore) e que normalmente exigem travessias completas sobre o mesmo. Alguns exemplos deste tipo de operações são:

- Revisão automática de fundos
- Inferência de valores (de níveis inferiores para níveis de topo)
- Contabilização de registos por nível de descrição
- Normalização da codificação (para resolver incongruências herdadas da retro-conversão)

A definição do catamorfismo fica a cargo da seguinte função:

```
Public Function Catamorphism(ByVal Gene As Gene) As Object
    Dim Child As LazyNode
    Dim ChildrenResults As New Collection
    For Each Child In Children()
        ChildrenResults.Add(Child.Catamorphism(Gene))
    Next
    Return Gene.Apply(Me, ChildrenResults)
End Function

Public Interface Gene
    Function Apply(ByVal obj As LazyNode, _
        ByVal ChildrenResults As Collection) As Object
End Interface
```

Exemplo de um Gene.

```
Public Class GeneValidator
    Implements Gene

    Public Function Apply(ByVal obj As LazyNode, _
            ByVal ChildrenResults As Collection) As Object

        Dim Result As ValidationErrorCollection
        Result = Validator.GetLazyNodeErrors(obj)
        Dim Child As ValidationErrorCollection
        For Each Child In ChildrenResults
            Result.Add(Child)
        Next
        Return Result
    End Function
End Class
```

O modelo adoptado é totalmente transparente no diz respeito à localização física da informação. Desde que os nove métodos descritos sejam correctamente implementados sobre a respectiva camada de dados, o mesmo conjunto de operações podem ser aplicadas sem mais alterações. A conversão entre modelos de dados é, também, simples e transparente.

Se pretendermos exportar uma árvore (fundo documental) armazenada numa base de dados relacional para um ficheiro EAD/XML basta efectuar uma travessia na árvore original e ir criando, progressivamente, os nós correspondentes na implementação XML do LazyNode.

Exemplo:

```
Private Sub MergeTrees(ByVal DstNode As LazyNode, ByVal SrcNode As LazyNode)
    Dim Index As Integer
    Dim ChildSrcNode As LazyNode
    Dim Children As LazyNodeCollection

    Children = DstNode.Children
    For Each ChildSrcNode In SrcNode.Children
        Index = Children.IndexOf(ChildSrcNode)
        If Index < 0 OrElse ChildSrcNode.isLeaf Then
            ' node doesn't exist. Create a new one
            Dim NewDstNode As LazyNode = DstNode.CreateNode()
           ' Copy all the stuff into the node
            NewDstNode.ImportFields(ChildSrcNode)
            NewDstNode.Upload()
            DstNode.AppendChild(NewDstNode)
            MergeTrees(NewDstNode, ChildSrcNode)
        Else ' Node exists... use that one
            Dim NewDstNode As LazyNode = Children.Item(Index)
            MergeTrees(NewDstNode, ChildSrcNode)
        End If
    Next
End Sub
```

Uma particularidade interessante deste modelo tem que ver com a sua independência ou portabilidade. Todas as operações descritas utilizam ferramentas *standard*, como o DOM ou o SQL, permitindo, assim, que o modelo seja facilmente transportado para outras plataformas.

O modelo de dados apresentado, embora simples de implementar e manipular, apresenta algumas desvantagens face a outras soluções, sendo a principal destas o facto de não tirar o máximo partido das capacidades oferecidas pelas bases de dados relacionais. A granularidade do modelo obriga a que toda a informação seja carregada registo a registo, ou seja, primeiro é carregada a raiz, e progressivamente, cada um dos nós filho (quando estes são necessários). A simples procura de um nó na árvore implica realizar uma travessia completa até o nó ser encontrado. Num modelo relacional puro, o registo seria encontrado pelo motor da base de dados, bastando-nos apenas executar uma query SQL. Este é o custo a pagar pela simplicidade e transparência do modelo, no entanto, os testes de utilizador efectuados ao sistema revelam que a informação é encontrada em tempo útil, ou seja, o utilizador não considera excessivo o de espera pela resposta do sistema, encarando o facto como normal.

6 Conclusões e trabalho futuro

Neste artigo é proposto um modelo de dados, baseado em *middleware*, que oferece simplicidade e transparência na manipulação de informação hierárquica residente em diferentes níveis lógicos de armazenamento. Foram criadas duas implementações concretas, uma sobre ficheiros XML baseados na norma EAD e outra sobre uma base de dados relacional. Em ambos os casos, a informação manipulada baseia-se num modelo hierárquico - uma árvore.

A passagem de informação de um modelo para o outro é assegurada por uma simples travessia. A aplicação de catamorfismos é conseguida indiferentemente do modelo utilizado.

O modelo apresentado, por ser abstracto e generalista, não tira partido das funcionalidades oferecidas pelas bases de dados relacionais. Assim, algum trabalho futuro poderia ser desenvolvido na optimização do modelo sobre bases de dados relacionais. A utilização de stored procedures seria um bom ponto de partida para concretização deste objectivo, tendo como principal desvantagem, a perda de independência e portabilidade, características do modelo proposto.

Outra possibilidade de optimização seria utilizar mecanismos de cache e/ou *pre-loading* de forma a minimizar o tráfego entre a aplicação e a base de dados. A utilização destes mecanismos acarreta informação de controlo adicional para garantir que a informação em cache se encontra coerente/actualizada.

Referências

[Bou] Ronald Bourret. Mapping dtds to databases. Internet.

[Day] Igor Dayen. Storing xml in relational databases. Internet.

[dNdD99] Comité de Normas de Descrição. *ISAD(G): Norma geral internacional de descrição arquivística*, second edition, Setembro 1999. Setembro.

[dRdBE03] Gabinete da Rede de Bibliotecas Escolares. A pesquisa de informação - o professor e a biblioteca, parceiros do aluno. Internet, Junho 2003.

[Fox00] Michael J. Fox. Ead cookbook, Julho 2000.

[Nun03] Rui Alexandre Nunes. Armazenamento de dados complexos utilizando xml. Master's thesis, Instituto Superior Técnico - Portugal, 2003.

[oA96] International Council on Archives. *International Archival Authority Record for Corporate Bodies, Persons and Families*. 1996.

[oAA02] Society of American Archivists. Ead standards home page. Internet, 2002.

[SL] Mustafa Atay Farshad Fotouhi Shiyong Lu, Yezhou Sun. A new inlining algorithm for mapping xml dtds to relational schemas. Technical report, Wayne State University - Department of Computer Science, Detroit, MI 48202.